Naval Research Laboratory

Washington, DC 20375-5320

NRL/MR/5540--00-8478

A Strategy of Security Services for Enterprise Applications

MYONG H. KANG
JUDITH N. FROSCHER

Center for High Assurance Computer Systems
Information Technology Division

JOON S. PARK

ITT Industries
Alexandria, VA

August 31, 2000

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 31, 2000	Final

4. TITLE AND SUBTITLE

A Strategy of Security Services for Enterprise Applications

5. FUNDING NUMBERS

6. AUTHOR(S)

Myong H. Kang, Joon S. Park,* and Judith N. Froscher

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5540--00-8478

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217-5660

National Security Agency
9800 Savage Raod
Ft Meade, MD 20775

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

*ITT Industries, Alexandria, VA 22303-1410

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

As the globalization of business becomes common practice, the need for secure enterprise computing increases. Even though many security solutions are available for enterprise computing today, they are, in general, designed to be applications independent. Therefore, each enterprise application has to adapt these solutions and tailor them for its specific use. In this paper, we investigate the security requirements for enterprise computing. We then present a strategy for providing solutions that can meet those requirements. Many of the requirements and solutions in this paper address the scalability of existing security solutions, the separation of enterprise application security from concrete organization level security enforcement, and the enforcement of fine-grained access control.

14. SUBJECT TERMS

Workflow security
Enterprise computing

15. NUMBER OF PAGES

19

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18
298-102

A Strategy of Security Services for Enterprise Applications

Myong H. Kang, Joon S. Park, and Judith N. Froscher
Information Technology Division
Naval Research Laboratory

{mkang, jpark, froscher}@itd.nrl.navy.mil

Abstract

As the globalization of business becomes common practice, the need for secure enterprise computing increases. Even though many security solutions are available for enterprise computing today, they are, in general, designed to be applications independent. Therefore, each enterprise application has to adapt these solutions and tailor them for its specific use. In this paper, we investigate the security requirements for enterprise computing. We then present a strategy for providing solutions that can meet those requirements. Many of the requirements and solutions in this paper address the scalability of existing security solutions, the separation of enterprise application security from concrete organization level security enforcement, and the enforcement of fine-grained access control.

1. Introduction

The Internet and business globalization have replaced the separation that was typical of the traditional business paradigm. Unconventional coalitions among businesses and nations are formed to advance common goals. These coalitions then quickly dissolve as individual objectives change. Threats now lie in these essential connections among participating enterprises, which also enable profitable cooperation. To facilitate these alliances, businesses and the military rely on distributed information technology (IT) for most operations. To support their missions, the enterprise needs

- Flexible IT resources and infrastructure that allow rapid configuration,
- Secure distributed applications that can be easily constructed across enterprise boundaries, and
- Enterprise-level anomaly detection and recovery.

Even though the above three requirements are equally important, we focus on the second item in this paper.

Building a flexible secure enterprise application is not a trivial task because a flexible secure enterprise application has to have proper structure so that

Manuscript approved July 14, 2000.

- Parts/components can be easily replaced,
- Application logic can be easily modified,
- Security policy can be easily adapted for different threat environments, and
- Different mechanisms for crossing enterprise boundaries can be supported.

We set up the following strategy to build flexible secure distributed applications.

- Make use/reuse of existing components and services as much as possible.
- Separate mission (meta) logic from component/service logic and develop an application-building tool that can program mission logic and specify interactions with underlying components/services. In general, components/services change less frequently. They are designed to perform predefined tasks or provide services to other programs. What does change is the mission (meta) logic that supports enterprise cooperation (i.e., information flow and control logic). Hence, this separation is very important for achieving flexibility in enterprise computing.
- Support abstraction. Since enterprise computing tends to be large scale, different designers and users tend to think about the problem at different levels of abstraction. Hence, enterprise-computing tools should facilitate this need so that users and application designers can design and think about the problem at levels at which they are comfortable.
- Maximum use of commercial-off-the-shelf (COTS) security solutions. There are many existing COTS security solutions for distributed computing, such as CORBA Security [1], Secure Socket Layer (SSL [2]), Role-Based Access Control (RBAC, [3]), etc. We make use of them as much as possible. Only when existing solutions do not provide the protection that is needed, do we extend them and incorporate the extended solutions into the application-building tool [4, 5, 6].
- Ensure that enterprise applications are interoperable to allow on-the-fly cooperation and recovery.

We have introduced a secure enterprise application-building tool [4, 5, 6]. This tool is based on the concept of enterprise application integration (EAI) and workflow management system (WFMS) tools. There are three major building blocks in the secure enterprise application-building tool as shown in Figure 1.

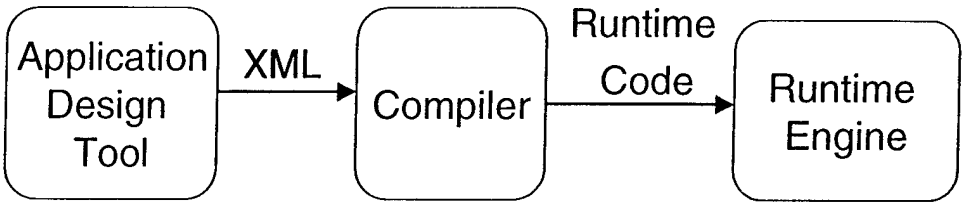

Figure 1: Internal structure of secure enterprise application building tool

The application design tool allows application designers to specify mission/application logic. In other words, the designer can specify each task (e.g., inputs, outputs, invocation method for the underlying component), and control logic and data flow among tasks. This tool also allows application designers to hide complexity by providing a way to group

related tasks into a higher-level task (the level of abstraction). The design tool saves the specification in XML (eXtensible Markup Language [7]). When the application design is completed, the compiler reads the XML representation of the design, and performs the necessary design analysis and validation. Finally, it generates runtime code for enactment services.

In this paper, our main security concerns are with the runtime engine. The following section summarizes the structure of the runtime engine.

2. Runtime Engine

Currently, we are using modified OrbWork [8] as our runtime engine. OrbWork is a single-level distributed workflow engine implemented in Java. It does not have a central scheduler; rather it is distributed with a scheduler per task. Each scheduler only knows its predecessors and successors.

Briefly, OrbWork consists of the following CORBA (Common Object Request Broker Architecture) servers: task servers, worklist servers, and data servers. Each task server may contain more than one task. Each task has three parts: task scheduler, task manager and the underlying component. The worklist server maintains the lists of pending work for human tasks. Data servers act as repositories for data that need to be accessed by tasks. Since they are CORBA servers, they communicate with each other through CORBA's IIOP (Internet Inter-ORB Protocol).

The task and worklist servers are not only CORBA servers but also HTTP (HyperText Transfer Protocol) servers. When a human operator has to interact with the worklist server (e.g., human task), he can do so through the HTTP protocol. Also when a human workflow manager needs to intervene for some reasons, he can do so through the HTTP protocol. Simplified communication paths among different components in OrbWork are shown in Figure 2.

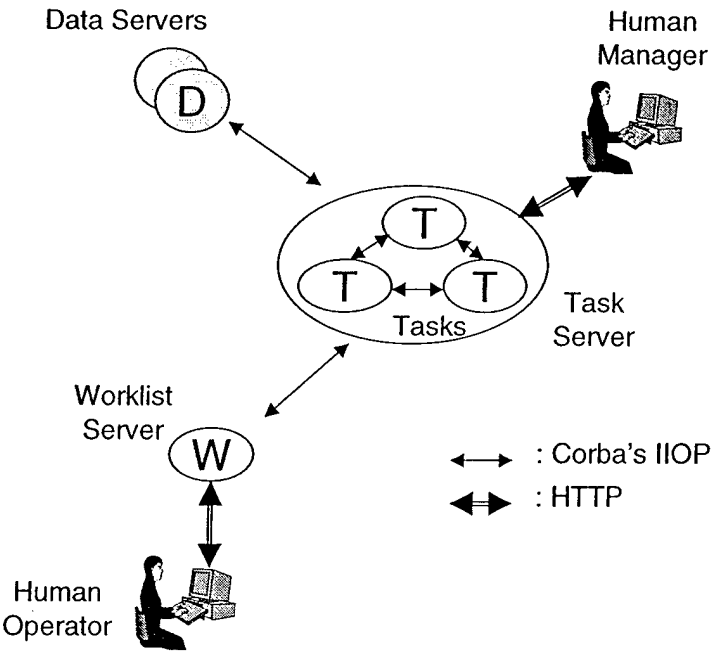

Figure 2: Communication paths among the components of runtime engine

3. Requirements for Application-level Security Services

In this section, we describe security requirements for enterprise computing. We explain why enterprise computing creates special security requirements and also focus on application-specific security requirements.

3.1. Secure Communication

From the brief description of the runtime engine in section 2, we can easily derive the following security requirements:

- Need to secure IIOP traffic
- Need to secure HTTP traffic

Securing IIOP and HTTP traffic is relatively straightforward by using industry standards, such as CORBA security and HTTPS [2] protocol. User identification and authentication (I&A) also can be achieved using existing identification and authentication mechanisms. However, there are several other enterprise application-specific security requirements, which require special consideration.

3.2. Separation of Enterprise Application Security Infrastructure from Organization Level Security Infrastructure

Enterprise applications have many unique features that general applications may not share. While general applications may be distributed, they are used by a single organization, which controls its own security administration. On the other hand, enterprise applications may span several organizations and must be flexible because organizations may join and leave the enterprise dynamically. Also an enterprise

application security policy must satisfy the autonomous security policies of member organizations and, at the same time, enforce enterprise constraints that make cooperation possible. Enterprise applications that support cooperation among several organizations may depend on the security infrastructure of the participating organizations. For example, an enterprise application may have to depend on several role servers from participating organizations to determine the privilege of participating users.

Because there are several organizations that participate in enterprise applications, the participants may change during the life cycle of an enterprise application. For example, a new organization may replace an old organization or there may be a merger or separation among organizations. Since each organization may support several enterprise applications, it is not realistic for each organization to restructure its security infrastructure for enterprise applications. Therefore, enterprise applications need to be insulated from organization level changes.

3.3. Providing Different Views of an Enterprise Application

An enterprise application that supports cooperation among several organizations may consist of tasks that are executed on many different hosts that are under the control of different organizations. Also in the case of human tasks, there are many different human operators who are authorized to execute them. Consider an enterprise application that supports cooperation among three organizations. It consists of six tasks where task B is under the control of organization 1, task A, E, and F are under the control of organization 2, and task C and D are under the control of organization 3 (see Figure 3).

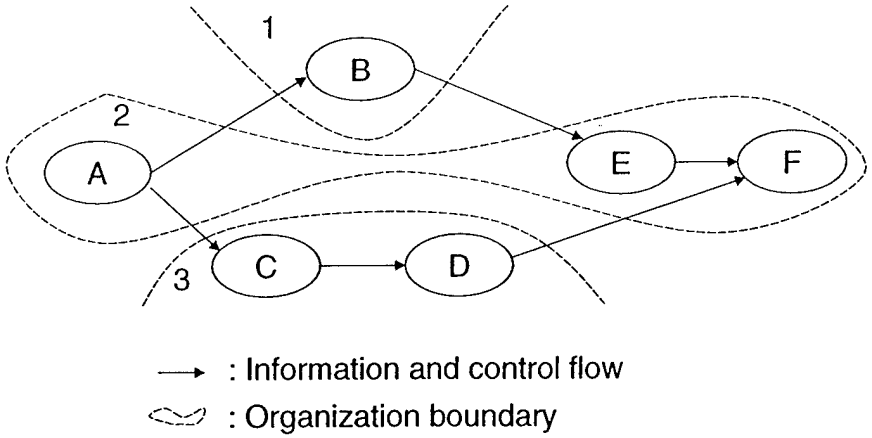

Figure 3: An example of enterprise application that supports cooperation among three organizations

From the above example, we can derive an extra security-related requirement for enterprise applications. We need to provide different views of the application based on users' needs-to-know. For example, users in organization 3 may not need to know the existence of other tasks such as task A, B, E, or F that are under the control of other organizations and even the existence of other tasks under the control of the same organization.

3.4. Dynamic Constraints

Dynamic constraints are required in many enterprise applications. Consider the following example: a simplified employee expense reimbursement scenario (see Figure 4). This example consists of five tasks; four human tasks and one automatic task. We assume that a required role is associated with each human task. Any human operator who has a role that is in the required role set or has more privilege than any of the roles in the required role set can execute the task. Note that the Issue_check and Sign_check tasks require the same role in this example. Throughout this paper, we use roles and RBAC examples because we believe RBAC can manage access control in large systems despite its limitations, which we will discuss later.

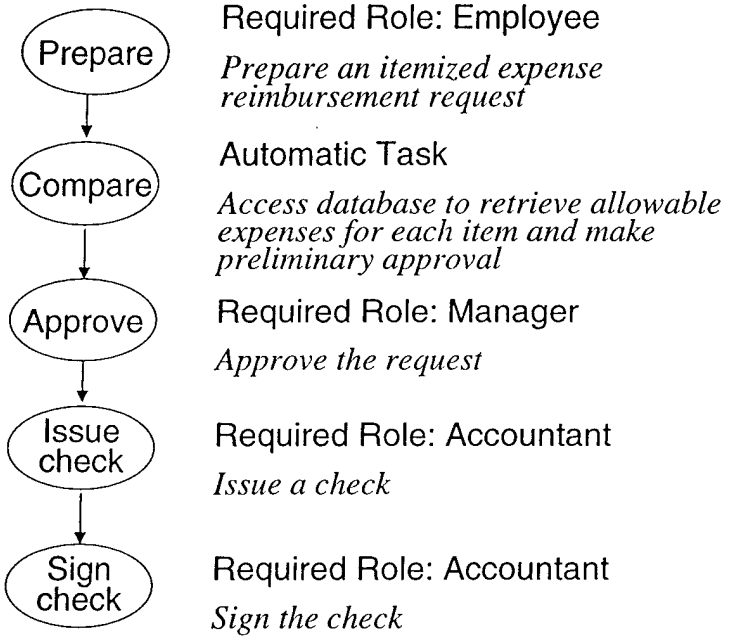

Figure 4: An example of a simplified employee expense reimbursement process

Consider a scenario where an employee prepares an expense reimbursement request. An organization may want to enforce a security policy that says the employee who prepared an expense reimbursement request should not approve the request. Thus we introduce dynamic constraints in RBAC in this particular example. This is a general application of the traditional 2-man rule (i.e., separation of duty [9]) that can be applied to two different tasks, Prepare and Approve, with two different required roles. In other words, if the employee who initiates the reimbursement process happens to be a manager then the manager should not approve the expense reimbursement request that he initiated even though he has both Employee and Manager roles. We can apply the 2-man rule to the two other tasks, Issue_check and Sign_check that have the same required role, Accountant. In this case, the 2-man rule says that a person who issues a check should not sign the check. This shows that the concept of dynamic constraints in a security model such as RBAC is relatively simple and clear. However, how to implement these constraints in a real system may not be a trivial task.

3.5. Fine-grained and Context-based Access Control

Enterprise applications tend to be large scale and consist of many components even within a process. For example, the task server in Figure 2 contains many tasks. Hence, conventional access control may be too coarse for enterprise applications. Traditionally, an access control decision is made based on subjects and objects. The subjects may be users or applications acting on behalf of users. The objects are data or resources in the system; hence, objects may be files in the file system. Conventionally, a process, which may be an application executing on behalf of a user, is the finest grained subject for which an access control decision can be made by the operating system (see Figure 5A). In Figure 5A, even if a user needs only permission[1] P1 to execute task T1, he will get P1, P2, P3, and P4.

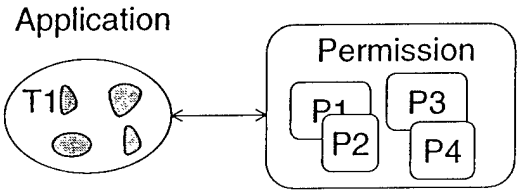

Figure 5A: Traditional access control model

As we mentioned earlier, enterprise applications may consist of many components. Hence, enterprise applications need access control based not only on an application but also on the components of the application (see Figure 5B). For instance, the task T1 acting on behalf of a user needs to have the permission P1. This scheme allows a user to get only permission P1 when he executes task T1.

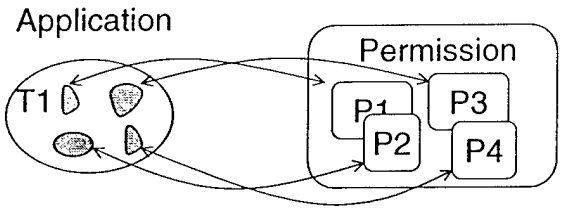

Figure 5B: Fine-grained access control model

We can refine the above example further. Consider a scenario where an application has a set of data objects (O_1, O_2, ... , O_n), which require permissions (P_1, P_2, ... , P_n) to be accessed respectively and all the permissions are assigned to role R. What if the application needs to permit an access to a subset of objects (e.g., O_1, O_5) to the user in a certain user's working context (e.g., when the user is working on task T3)? One may

[1] Permission is a set of authorized interactions that a subject can have with one or more objects in the system. Permission, especially in the RBAC model, may have a variety of interpretations. For instance, it can be applied to whole organizations, objects, or particular fields in an object. Actually, permissions defined in a conceptual model (Figure 5A & B) can be implemented in many ways in different systems.

create separate roles corresponding to each permission. However, this is not a good solution, because the constraints are dynamic and application-specific. Furthermore, the number of permissions may potentially be quite large if there are several enterprise applications. Also if an application has too much flexibility (e.g., creating roles for each permission and defining the relationships between roles) to support fine-grained access control, it is very hard to maintain a global view of the organization. This demonstrates a shortcoming in the conventional RBAC approach.

An example clarifies the need for context-based access control. Consider Issue_check and Sign_check tasks in Figure 4. Assume that the check data object has three fields; amount, payable_to and signature. Issue_check task can modify only amount and payable_to in the check data object. But Sign_check task can modify only signature; all other data fields should be read-only to this task. Therefore, even though the two tasks have the same required role (i.e., Accountant), the access control requirements may be different. This kind of fine-grained data access policy can be relatively easily enforced by the application but is rather difficult to enforce through typical RBAC mechanisms (see Figure 6).

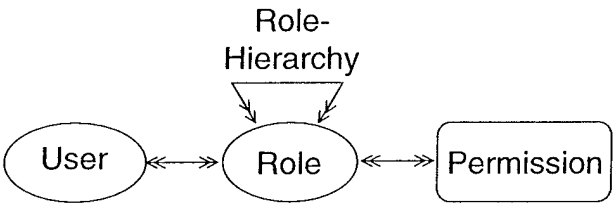

Figure 6: A simplified RBAC model

The reason for these difficulties in RBAC is that permissions are simply assigned to the user by means of the user's roles regardless of the resource-accessing context. In the above example, Issue_check and Sign_check tasks have the same required role, Accountant. However, we need different permissions based on the context in which a user is working.

In summary, enterprise applications have many additional application-specific security requirements as we described. We introduce the components that are needed for secure enterprise computing in section 4. In section 5, we provide strategies to satisfy the security requirements that we have introduced in this section.

4. Enterprise System Model

There are five major components in our enterprise computing models: policy editor, policy server, runtime engine, monitor, and users. We introduce two different operational models for RBAC; user-pull and server-pull models [10]. Figure 7 shows the components and their relationships in the models. The main difference between the two models is who pulls the user's role information from the policy server.

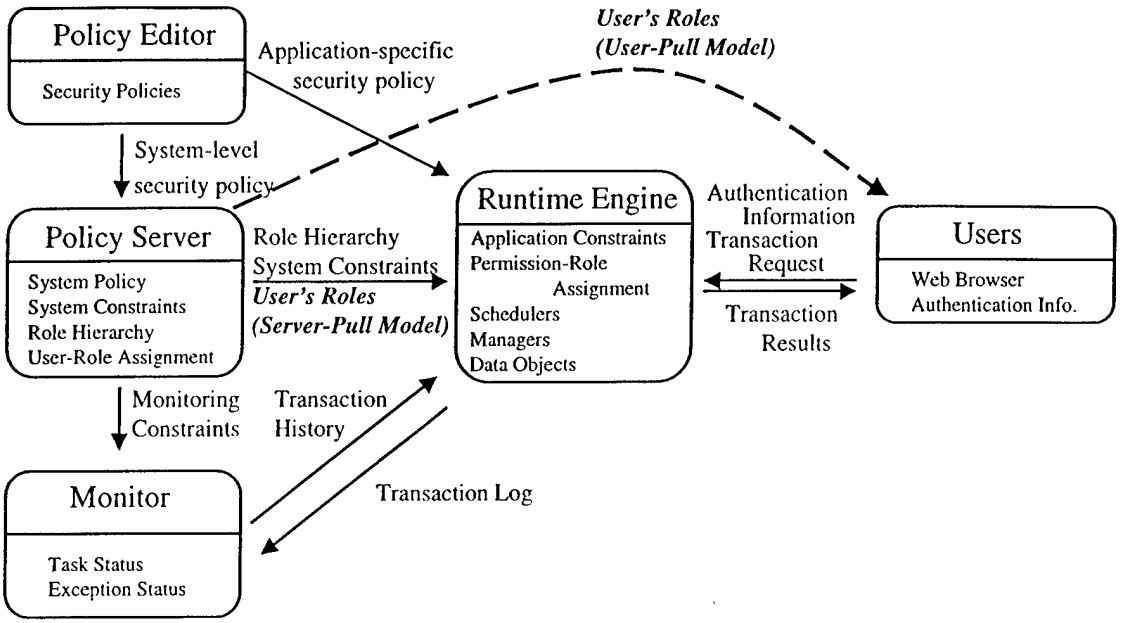

Figure 7: A system model for enterprise computing

The ***policy editor*** allows application designers to express their system and application-specific security policies. For example, an application designer may specify 2-man rule, data access policy, or role hierarchy [3] though this editor. System-level policy (e.g., role hierarchy) is loaded to the policy server and application-specific policy (e.g., 2-man rule) is transferred to the runtime engine.

The ***policy server*** provides system policy and system-level constraints to the runtime engine and monitor. For instance, it defines authentication mechanisms and resource availability for each application in the system. Typical components of the policy server may be a role server and certificate authority. The role server provides role hierarchy and user-role assignment information to support RBAC. The certificate server may support PKI (Public Key Infrastructure) in the enterprise environment. It is possible to have a separate role server or certificate authority in each organization. It is important to note that the policy server does not have application-specific constraints. In other words, application-specific constraints are defined though the policy editor and enforced by the runtime engine in our approach.

The ***runtime engine*** consists of runtime codes and specification generated by the design tool (see Figure 1). During installation and execution, the runtime engine refers to the policy server for system-level constraints, such as resource usage, authentication mechanisms, user-role assignment and role hierarchy. Providing a secure distributed computing environment is one of our primary objectives. Therefore, the runtime engine has to enforce application-specific security constraints (e.g., 2-man rule), access control (e.g., RBAC), and other application-specific constraints (e.g., timeout).

The ***monitor*** consists of a monitor server and client. The monitor server receives event information from the runtime engine and records it. The monitor server has application-

layer monitoring functions that provide event information, based on its clients' interests. Furthermore, the monitor server provides the transaction history to the runtime engine (if it is necessary) so that the runtime engine can make the correct decision that complies with the security constraints based on the user's previous transaction history [11] (e.g., 2-man rule). The monitor client registers its topics of interest (e.g., only events during a certain period of time, only events related to a task abortion) to the monitor server. If the request does not violate the monitoring constraints, which are provided by the policy server, these topics will be sent to the monitor clients. In other words, different monitor clients may have different interests. For instance, monitor client M1 may register for exception status ES and task status TS of task T1 and T2 in application A1 by sending a message [M1: {application, (list of tasks), (list of topics)}, {A1, (T1, T2), (ES, TS)}] to a monitor server. After the registration, the monitor server pushes the information related to M1's registered topics to the monitor client M1.

In our case, ***users*** communicate with the runtime engine using Web browsers via HTTP or HTTPS, because task schedulers are Web servers. Users are assigned roles under RBAC in the policy server. When a user connects to the runtime engine, the runtime engine authenticates the user and retrieves the user's role information from the user's credential (user-pull model) or from the policy server (server-pull model). We assumed that each user already has his or her authentication credential in the system by means of existing authentication mechanisms such as passwords, Kerberos, X.509, and so on. Alternatively, it is possible for each user to receive his or her authentication credential and roles from the policy server in a single entity [10] and use it in the runtime engine. For instance, if we use X.509 certificates, the policy server can add the user's roles to the user's certificate, which has the user's public-key information. As a result, a single user-certificate can be used for both authentication and authorization. However, we do not claim that this kind of bundled certificate is always good. Especially, if the lifetime of a user's role and public-key information are different, or if different authorities must issue the role and identity information, bundled certificate may not be a good solution. Instead, we can use two different certificates to satisfy the above requirements. In this case, we must support the binding of attributes and identification for each user. For instance, if Alice presents Bob's attributes with her authentication information to the Web server, she must be rejected.

We introduced two ways to access users' role information: user-pull and server-pull. In the user-pull model, the user, let's say Alice, pulls her roles (see dotted arc in Figure 7) from the role server, which is a part of the policy server in our model, and then presents the role information to the runtime engine along with her authentication information. In the server-pull model, the user presents her authentication information to the runtime engine. After a successful authentication, the runtime engine pulls the user's role information from the role server for RBAC.

The user-pull model requires a user's cooperation to obtain her roles, but it enhances the runtime engine performance. Once the user obtains her roles, she can use them in many different sessions until the roles expire, which increases the attribute reusability. However, the longevity of the role decreases the freshness of the attributes. Thus the

policy server should push the status change of user's roles, such as role revocation, to the runtime engine during runtime for updated information.

The server-pull model requires the runtime engine's cooperation for obtaining the user's role information – which decreases the runtime engine performance - from the policy server. In this model, the runtime engine retrieves the user's role information from the policy server for each session. This increases the freshness of the attributes, so the information update (e.g., role revocation) is more efficient than user-pull model. However, it decreases attribute reusability and increases the single-point failure vulnerability because every session requires an access to the policy server. We summarize the pros and cons of the two approaches in Table 1.

Table 1: User-pull vs. server-pull models.

Operational Models	User-Pull	Server-Pull
User's Convenience	Low	High
Runtime Engine Performance	High	Low
Reusability	High	Low
Attribute Freshness	Low	High
Single-Point Failure	Low	High

5. A Strategy for Application-specific Security Services

As we mentioned in section 1, one of our strategies for secure enterprise computing is the maximum use of unmodified COTS security solutions whenever they can be utilized. We already mentioned PKI, CORBA security and RBAC as examples of COTS security solutions that are useful for secure enterprise computing. However, these security solutions were developed independent of applications. Each enterprise application needs to tailor them to satisfy its own security requirements. We presented a few application-specific security requirements for enterprise computing in section 3. In this section, we focus on security mechanisms that can satisfy the application-specific security requirements based on the models that we have introduced in section 4.

5.1. Role Domain and Policy Server

RBAC is a convenient way for a system administrator to create roles, grant permissions to the roles, and assign users to the roles on the basis of their job responsibilities and the system policy. Because there may be many organizations that are involved in the enterprise cooperation, an enterprise application may span several role servers. Therefore, if there is a change in participating organizations, the application has to be changed. To avoid such disruptions, we need to decouple the enterprise application security infrastructure from organization level security infrastructures.

In general, a role server has organization-specific role structures that specify available roles and role hierarchy, and user-role assignments in the organization. If an enterprise application accesses the organization's role server directly, we cannot achieve this decoupling between the enterprise application security infrastructure and organization

level security infrastructure. To achieve this separation, we introduce a *role domain* which is a role structure interface for an enterprise application. The role domain information resides in the policy server (see Figure 7). An enterprise application accesses the role structure that was provided by role domains. Any organization that needs to participate in the cooperation should map its role structure to the role domain of an enterprise application.

Consider the enterprise application in Figure 3. The application may be designed to interact with three role domains, where there is one role domain per organization. In this case, the mapping from the role structure of an organization to that of the role domain will be straightforward. We can think of another case where the application was designed and being used without knowing that there are three participating organizations (i.e., the changes in participating organizations should not affect the application). In this case, the application may be designed to interact with only one common role domain. Again, it is each participating organization's responsibility to map its role structure into the role domain. Of course we can think of hybrid cases, as well.

The relationship between a role domain and the role structures of organizations is similar to an interface in client-server interactions. In a distributed client-server application, the client software accesses a server implementation through the interface. Even though a server implementation may change, it does not affect the client as long as the server implements the interface correctly. Enterprise applications access role information through role domains and each organization provides a concrete role structure. By decoupling organization-specific security enforcement mechanisms from enterprise applications, we reduce unnecessary modification of applications. This is also a convenient way to extend RBAC for enterprise computing.

We provide a way to specify required role domain and role information for each task in an enterprise application in the policy editor (see Figure 7). Application designers' assignment of roles to permissions is turned into an access control policy that each task will enforce. For example, each human task has a required role set,

$$[\{\text{roleDomain, (roles)}\}, \{\text{RD\_1, (A, B, C, ...)}\}, \ldots]$$

where RD_1 is a specific role domain and A,B,C are specific roles in role domain RD_1. In this example, if a user belongs to one role domain in the required role set and has one of the roles in the role domain or more privilege than one of the roles in the required role set, he is allowed to perform the task.

If an enterprise application was designed to have one role domain, then there may be only one common policy server for the application. This common policy server provides convenience and performance for both application designers and the runtime engine. However, the synchronization between the common policy server and the other policy servers that are maintained by participating organizations may not be a trivial task. If we assume that different role domains are administered by different organizations, it is likely that each organization has its own policy server. In this case, application designers may have to access several policy servers to obtain role information during design time, and

the runtime engine itself has to access several policy servers during runtime to make an access control decision.

5.2. Providing Different Views of an Application

Enterprise applications are complex with many components. Therefore, we need to provide different views of the application to different users. For example, a user may think the application consists of only components C1, C2, and C3. Another user who has more privilege than the previous user may think that the application consists of components, C1, C2, C3, and C5.

Providing different views of an application to different users is fairly trivial. Since a required role set is associated with each task in our approach, the application can hide the existence of certain tasks unless the user has the proper privilege (e.g., role) for the tasks. Consider an application that consists of ten tasks. If a user, whose role is D in role domain RD_1, accesses this application and the user has privileges for only five tasks, then this application appears to consist of five tasks to this user. In other words, the knowledge of the existence of a task, let alone access to the task, is denied unless the user has the privileges for accessing the task.

5.3. History-based Access Control for Dynamic Constraints

We already mentioned that application-independent security solutions might not provide a satisfactory solution for enterprise applications. For example, an application-specific security policy, which says an accountant, who performs Issue_check, shall not execute Sign_check on the same data object (see Figure 4), may not be easily enforced by application independent security solutions. In general, dynamic constraints, such as dynamic separation of duty (DSD) that we described in the previous example, are required by enterprise applications.

Several RBAC models have been proposed and implemented for efficient and strong access control. However, most approaches focus on the mechanisms between users and their available roles to support dynamic constraints. For instance, NIST used active role sets (ARS) in their RBAC/Web implementation [12] to support DSD. In this implementation, when a user is assigned authorized roles, he can activate any roles among his authorized roles except the roles in a DSD relationships (if any). Thereafter, the user is allowed to use the permissions assigned to the roles in his active role set (see Figure 8). However, any pair of roles in ARS cannot have a DSD relationship.

In general, the responsibility of user-role management (including creating or removing users, roles, user-role assignment, role relationships, role constraints, role hierarchy and global constraints) belongs to each organization. A role server of an organization may maintain this information and this information usually reflects the structure of an organization (e.g., job responsibility within organization). This implies that the global role constraints are applied to all applications in the organization. If each application requires different application-specific dynamic constraints, there may be conflicting constraints from different applications. To support the dynamic separation of duty constraint, we might use the ARS approach. However, if two tasks require different roles,

a malicious user may perform the two tasks – which are in a separation of duty relationship – by activating different ARSs. Furthermore, if the two tasks require the same role, the ARS approach may not provide satisfactory solutions for enterprise computing, since RBAC does not provide sophisticated access control for a sequence of events.

To overcome some of these difficulties, we propose to use history-based access control [11]. We introduced the interactive monitor server (see section 4) to support this mechanism. The monitor has application-layer monitoring functions based on its interests in the application, and provides the transaction history to the applications for history-based access control decisions during runtime. Access control is based on the user's activity history as well as his roles. When a user accesses a task, the user has to present a credential that includes his role, role domain, and identity. When a user is allowed to access a task based on his credential, that event is logged to the monitor server. If a task has dynamic constraints, it will check the event history before it allows user access. Our policy editor (see Figure 7) allows an application designer to specify dynamic constraints for each task. For example, if task3 has a dynamic constraint, !Performer(task1), which means if a user performed task1 then the user cannot perform this task, then task3 will check the event history before it allows the user to execute task3.

The philosophy of our solution is as follows. Since dynamic constraints, in general, are related to the semantics of applications, application independent mechanisms cannot provide satisfactory solutions. It is conceptually clearer to associate them with applications and provide mechanisms in the application for their enforcement. These mechanisms are not specific to any one application but instead are mechanisms that secure applications must provide to satisfy security requirements for enterprise computing.

5.4. Fine-grained and Context-based Access Control

In a typical RBAC model, a set of permissions (e.g., access right) is associated with a role. Therefore, the application can enforce access control based on role information. For example, we can attach an access control policy on data type definitions (DTD) of XML representations of data [13] or on an interface definition language (IDL). However, such approaches cannot provide fine-grained data access control, because DTD or IDL applies to the whole application.

Typical security solutions provide access control at the granularity of a file or process, in general. However, if there are many tasks in a process as we mentioned earlier (section 3.5), the access control of a task becomes rather difficult. Consider an enterprise application that accesses 10 data objects. Among the ten objects, task1 can access only three data objects. However, task1 may not access all the fields in the three data objects. What we need is fine-grained data access control that is based not only on the role of a user but also the task that the user is working on.

To support such fine-grained access control, we introduce a new approach in RBAC. Even though we apply this idea to RBAC here, it can be used for any system that requires

a strong and efficient access control mechanism. We believe that user-role management should be organization specific while role-permission management should be application-specific in a large enterprise-computing environment. Different applications may have different permissions sets and constraints, and the role-permission assignment could vary in different applications. Under this circumstance, if we apply the application-specific constraints between roles and permissions in each application, we can provide the fine-grained and context-based access control in each application without losing global consistency.

In this approach, the access decision for the task is determined by the required role set. The permission is granted to the user based on his task context (i.e., a user working on the specific task). Figure 8 shows how conventional ARS approach described in section 5.3 and our context-based approach are applied in the RBAC model.

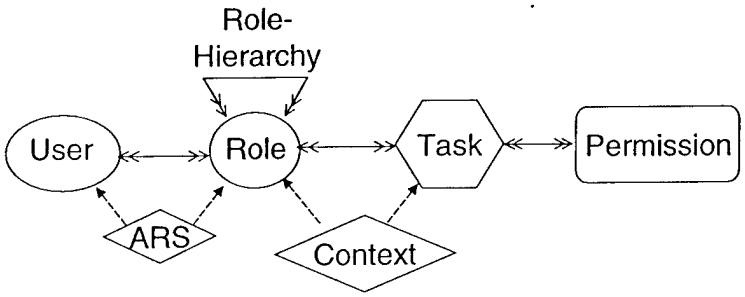

Figure 8: Comparison of ARS and context-based RBAC

In this way, we can provide fine-grained access control mechanisms to the applications. In our previous example in section 3.5, an application is able to grant access to only a subset of the objects (O_1, O_5) in a certain context to a user, who has role R. Additionally, this approach provides more convenience to users, because it is transparent to the users. In contrast, the ARS approach requires a user's cooperation. However, it is also possible for applications to use both ARS and our context-based access control mechanisms.

We associate a data access policy with each task in an enterprise application. Our policy editor (see Figure 7) provides a convenient way to specify the data access policy for each task. This data access policy describes which fields of which data object can be accessed by a specific task. In other words, each task has its associated data access policy that has a series of the following triples,

{Data object, field name, permission}

where permission can either be read-only, full-control, no-access, etc. In addition to a data access policy, human tasks have a required role set that we described earlier in section 5.1. Therefore, a task determines which user in which role can access the task. Once a user has the correct required role as we discussed earlier, the user is allowed to access the task. Furthermore, data access is further restricted by the task's context. Figure 9 shows that Task1 can access only three data objects and accessing the fields of those data objects is further restricted by a task-specific policy.

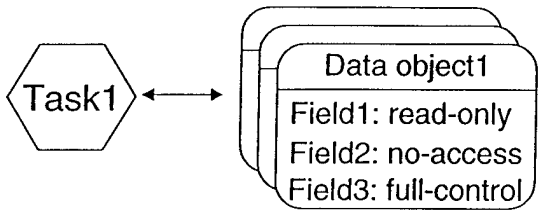

Figure 9: Task-specific data access control

It is interesting to compare the context-based access control that was introduced in this section to TBAC [14]. In TBAC, permissions are granted for a limited period of time based on each task. The permissions may be revoked after the time period elapses. In general, the execution (activation) order of tasks can be controlled in an enterprise application; however, the execution time of each task (i.e., the order of task execution) cannot be easily controlled especially when the two tasks are from different applications. Therefore, if the access permission is based on time periods, unnecessary constraints or undesirable permissions may be introduced. Context-based access control does not introduce unnecessary constraints across applications because permissions are tied to each task, thus permissions are managed in a distributed fashion.

6. Conclusion

In this paper, we studied enterprise application-specific security requirements such as dynamic constraints, fine-grained access control, and the need to insulate enterprise applications from organization level changes. Most of the existing security solutions do not satisfy the security requirements of enterprise applications. In this paper, we presented a strategy for modifying and extending existing solutions to satisfy the security requirements of enterprise applications. We have introduced the role domain as an interface between enterprise applications and organization-specific security infrastructure. We also have introduced history-based access control for dynamic constraints, fine-grained and context-based access control. A detailed implementation description of those mechanisms can be found in [15]

References

1. "CORBA Security Service Specification," Object Management Group (OMG), December 1998.

2. D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol," In Proceedings of the Second UNIX Workshop on Electronic Commerce, November 1996.

3. R. S. Sandhu, E. J, Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," IEEE Computer, 29(2): 38-47, February 1996.

4. M. H. Kang, J. N. Froscher, B. J. Eppinger, and I. S. Moskowitz, "A Strategy for an MLS Workflow Management System" In Proceedings of 13th IFIP Conference on Database Security, Seattle, WA, 1999.

5. M. H. Kang, J. Froscher, and B. Eppinger, "Toward an Infrastructure for MLS Distributed Computing," In Proceedings of 14th Annual Computer Security Applications Conference, Scottsdale, AZ, 1998.

6. M. H. Kang, B. J. Eppinger, and J. N. Froscher, "Tools to Support Secure Enterprise Computing," In Proceedings of 15th Annual Computer Security Applications Conference, Phoenix, Arizona, December 1999.

7. "Extensible Markup Language (XML) 1.0," World-wide-Web Consortium, http://www.w3.org/TR/1998/REC-xml-19980210.html.

8. K. Kochut, A. Sheth, and J. Miller, "ORBWork: A CORBA-Based Fully Distributed, Scalable and Dynamic Workflow Enactment Service for METEOR," UGA-CS-TR-98-006, Technical Report, Department of Computer Science, University of Georgia, 1998.

9. R. Simon and M. E. Zurko, "Separation of Duty in Role-Based Access Control Environments," In Proceedings of New Security Paradigms Workshop, September 1997.

10. J. S. Park and R. Sandhu, "Smart Certificates: Extending X.509 for Secure Attribute Services on the Web," In Proceedings of 22nd National Information Systems Security Conference, Crystal City, VA, October 1999.

11. G. Edjlali, A. Acharya, and V. Chaudhary, "History-based Access Control for Mobile Code," In Proceedings of the 5th ACM Conference on Computer and Communications Security, San Francisco, CA, November 1998.

12. D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn, "A Role Based Access Control Model and Reference Implementation within a Corporate Intranet," ACM Transactions on Information Systems Security, Volume 1, Number 2, February 1999.

13. D. L. Long, J. Baker, and F. Fung, "A Prototype Secure Workflow Server," In Proceedings of 15th Annual Computer Security Applications Conference, Phoenix, Arizona, December 1999.

14. R. K. Thomas and R. S. Sandhu, "Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management," In Proceedings of the IFIP WG11.3 Workshop on Database Security, August 1997.

15. M. H. Kang, "An Approach for Secure Enterprise Applications," In preparation.